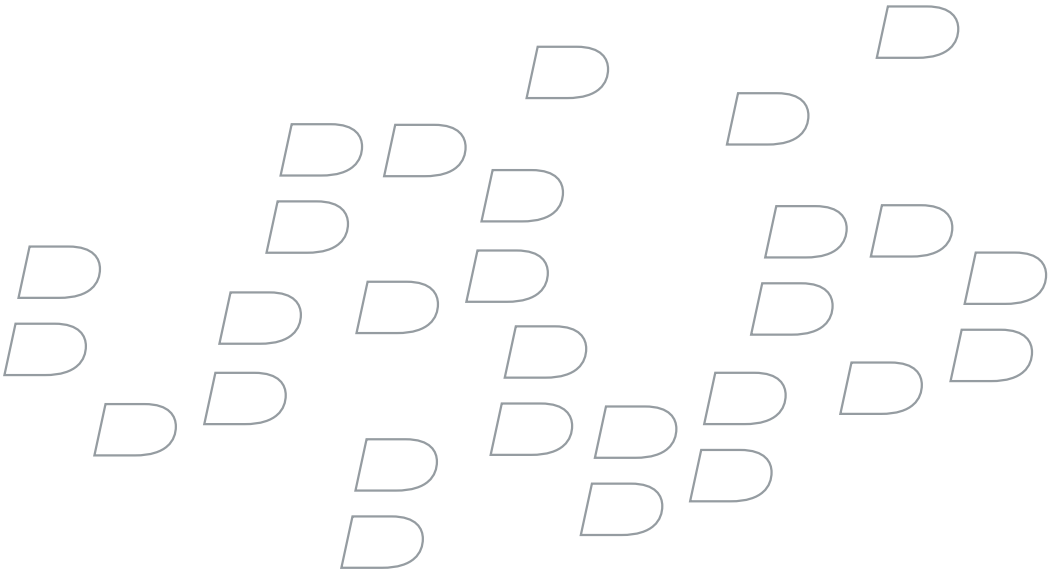

BlackBerry Java Development Environment

Version 4.3.0

Cryptographic Smart Card Driver Development Guide

BlackBerry Java Development Environment Version 4.3.0 Cryptographic Smart Card Driver Development Guide

Last modified: 6 September 2007

Part number: 12802084

At the time of publication, this documentation is based on the BlackBerry Java Development Environment Version 4.3.0.

Send us your comments on product documentation: https://www.blackberry.com/DocsFeedback.

©2007 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry, "Always On, Always Connected" and the "envelope in motion" symbol are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

Bluetooth is a trademark of Bluetooth SIG. Java is a trademark of Sun Microsystems, Inc. SafeNet is a trademark of SafeNet, Inc. Casira is a trademark of Cambridge Silicon Radio Ltd. RSA is a trademark of RSA Security. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The BlackBerry device and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit www.rim.com/patents for a list of RIM [as hereinafter defined] patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical or other inaccuracies in this document. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third party sources of information, hardware or software, products or services and/or third party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third Party Information or the third party in any way. Installation and use of Third Party Information with RIM's products and services may require one or more patent, trademark or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses relating to Third Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third Party Information shall be governed by and subject to you agreeing to the terms of the Third Party Information licenses. Any Third Party Information that is provided with RIM's products and services is provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the Third Party Information and RIM assumes no liability whatsoever in relation to the Third Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages.

Certain features outlined in this document require a minimum version of BlackBerry Enterprise Server Software, BlackBerry Desktop Software, and/or BlackBerry Handheld Software and may require additional development or third-party products and/or services for access to corporate applications.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House, 36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada

Contents

1 Using smart cards 3
Smart cards 3
Add support for smart cards 3
Creating a cryptographic smart card driver 4
Set up the project for the cryptographic smart card driver 4
Design a cryptographic smart card driver 4
Create a cryptographic session for a cryptographic smart card 5
Create a cryptographic token for private key operations 6
Store the location of the private key file 8

2 Testing a cryptographic smart card driver 11
Set up the BlackBerry Device Simulator to test a cryptographic smart card driver 11
Set up a BlackBerry device to test a cryptographic smart card driver 11
Test the cryptographic smart card driver 12

3 Code samples 15
Code sample: Creating a cryptographic smart card object 15
Code sample: Creating a cryptographic session for a cryptographic smart card driver 18
Code sample: Enabling a CryptoToken object for RSA operations using a private key 24
Code sample: Storing the location of a private key file on the smart card 28

1

Using smart cards

Smart cards
Creating a cryptographic smart card driver

Smart cards

Add support for smart cards

Smart card scenarios	Description
supported smart cards	The BlackBerry® device supports the following smart cards: • Common Access Card (CAC) • SafeNet® Model 330 smart card
unsupported smart cards	If your smart card is not a CAC or SafeNet smart card, use the smart card API to create a cryptographic smart card driver to support your smart card.

You can use the `net.rim.device.api.smartcard` package and the `net.rim.device.api.crypto` package to create smart card drivers that interact with smart cards and BlackBerry Smart Card Readers. The smart card API was included in BlackBerry® Java® Development Environment Version 4.1. The `SmartCardProtocolErrorException` API was added in BlackBerry JDE Version 4.2.

A cryptographic smart card driver that implements the smart card API can work with the S/MIME Support Package for BlackBerry® smartphones on a BlackBerry smartphone with S/MIME support. A cryptographic smart card driver can perform private key operations on the smart card such as signing and decrypting messages. A cryptographic smart card driver does not require the S/MIME Support Package for BlackBerry smartphones to be able to import certificates from the smart card, or to provide two-factor authentication for a BlackBerry device. See *BlackBerry with the S/MIME Support Package* for more information on S/MIME.

The smart card API information included with BlackBerry JDE Version 4.2 or later contains some deprecated elements. The deprecated elements provide backward compatibility for a cryptographic smart card driver application created for a BlackBerry device that uses BlackBerry® Device Software Version 4.1.x. If you want to create a cryptographic smart card driver for a BlackBerry device that is compatible with either BlackBerry Device Software Version 4.1.x or Version 4.2 or later, you can use the deprecated elements to avoid having to create two versions of the cryptographic smart card driver.

If you want to create a cryptographic smart card driver for a BlackBerry device that is compatible with BlackBerry Device Software Version 4.2 or later, use the non-deprecated API items in the smart card API.

Creating a cryptographic smart card driver

To create a cryptographic smart card driver for BlackBerry Device Software Version 4.1 or later, complete the following tasks:

1. Set up the project for the cryptographic smart card driver.
2. Design a cryptographic smart card driver.
3. Create a cryptographic session for a cryptographic smart card driver.
4. Create a cryptographic token for private key operations.
5. Store the location of the private key file.

Set up the project for the cryptographic smart card driver

Task	Steps
Set up the project.	1. Open the BlackBerry® Integrated Development Environment. 2. Create a new project for the cryptographic smart card driver.
Configure the project.	1. In the Workspace window, right-click the new project. 2. Select **Properties**. 3. On the **Application** tab, in the **Project type** field, type **Library**. 4. Select the **Auto-run on startup** check box. 5. In the **Startup Tier** field, select the **7(Last; 3rd party apps only)** option. 6. On the **Build** tab, in the **Imported jar files** field, add the **net_rim_crypto.jar** file. 7. Click **OK**.
Create a `libMain()` method.	> In your extension of the `CryptoSmartCard` class, implement the `libMain()` method.

Design a cryptographic smart card driver

You must implement all of the following methods:

Task	Steps
Enable a smart card driver to open a cryptographic session with a smart card.	> Implement `SmartCard.openSessionImpl(SmartCardReaderSession)`.
Enable a smart card driver to verify if a smart card is compatible with a specific Answer To Reset (ATR) sequence.	> Implement `SmartCard.checkAnswerToResetImpl(AnswerToReset)`.
Enable a smart card driver to display settings or properties.	> Implement `SmartCard.displaySettingsImpl(Object)`.
Enable a smart card driver to indicate support for display settings.	> Implement `SmartCard.isDisplaySettingsAvailableImpl(Object)`.
Retrieve the capabilities of a smart card.	> Implement `SmartCard.getCapabilitiesImpl()`. The capabilites of a smart card include the protocols the card supports, the baud rate, and the clock adjustment factors.
Retrieve the smart card type.	> Implement `SmartCard.getLabelImpl()`.

Task	Steps
Retrieve the names of the algorithms that the smart card supports, for example ("RSA", "DSA").	> Implement `CryptoSmartCard.getAlgorithms()`.
Retrieve a `CryptoToken` object that supports the given algorithm.	> Implement `CryptoSmartCard.getCryptoToken(String)`.

See "Code sample: Creating a cryptographic smart card object" on page 15 for more information.

Create a cryptographic session for a cryptographic smart card

Task	Steps
Create a cryptographic smart card session for a cryptographic smart card.	> Extend the abstract `CryptoSmartCardSession` class.
Close a cryptographic smart card session.	> Implement `SmartCardSession.closeImpl()`.
Retrieve the maximum number of login attempts.	> Implement `SmartCardSession.getMaxLoginAttemptsImpl()`.
Retrieve the ID for the smart card.	> Implement `SmartCardSession.getSmartCardIDImpl()`.
Retrieve the remaining number of login attempts.	> Implement `SmartCardSession.getRemainingLoginAttemptsImpl()`.
Attempt to log in to the cryptographic session using a given password string.	> Implement `SmartCardSession.loginImpl(String)`.

Retrieve the certificates from the smart card.	> Perform one of the following steps:
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, implement `CryptoSmartCardSession.getKeyStoreDataArrayImpl()` as follows:
	`RSACryptoToken token = new MyRSACryptoToken();`
	`RSACryptoSystem cryptoSystem = new RSACryptoSystem(token, 1024);`
	`RSAPrivateKey privateKey;`
	`PrivateKey privateKey = new RSAPrivateKey(cryptoSystem, new MyCryptoTokenData());`
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, implement the `getKeyStoreDataArrayImp` method as follows:
	`PrivateKey privateKey = CryptoSmartCardUtilities2.createPrivateKey(token, 1024, new MyCryptoTokenData());`

Validate the input parameters in the cryptographic session.	> Perform one of the following steps:
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, in your implementation of the `signDecrypt` method, validate the parameters as follows:
	`int modulusLength = cryptoSystem.getModulusLength();`
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, in your implementation of the `signDecrypt` method, validate the parameters as follows:
	`int modulusLength = (cryptoSystem.getBitLength()/8);`

Task	Steps
Retrieve random data from the internal random number generator of the smart card.	> Implement `CryptoSmartCardSession.getRandomBytesImpl(int maxNumBytes)`.

See "Code sample: Creating a cryptographic session for a cryptographic smart card driver" on page 18 for more information.

Create a cryptographic token for private key operations

Task	Steps

Create a token class.	> Perform one of the following steps:
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, create a class that extends an RSA®, DSA, or ECC token class. For example:
	`final class MyRSACryptoToken extends RSACryptoToken implements Persistable`
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, create a class that extends the `SmartCardRSACryptoToken` class.
	`final class MyRSACryptoToken extends SmartCardRSACryptoToken`

Determine if the token object can perform authentication for a BlackBerry device user.	> Create a method that returns true if your token object prompts the BlackBerry device user for authentication information.
	`public boolean providesUserAuthentication()`
	`{`
	`return true;`
	`}`

Determine if the token object supports the current `CryptoSystem`.	> Create a method that returns a Boolean value that indicates if the token object supports the current `CryptoSystem`.
	`public boolean isSupported(CryptoSystem cryptoSystem, int operation)`
	`{`
	`return (operation == PRIVATE_KEY_OPERATION);`
	`}`

Task	Steps

Determine if the token object and the `CryptoSystem` support the type of encryption scheme.	> Perform one of the following steps:
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, create a method that returns a Boolean value that indicates if the token object supports the specified encryption scheme.
	`public boolean isSupportedDecryptRSA(RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData)throws CryptoTokenException`
	`{`
	`return privateKeyData instanceof MyCryptoTokenData;`
	`}`
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, create a method that returns a Boolean value that indicates if the token object supports the specified encryption scheme.
	`public boolean isSupportedDecryptRSASmartCardImpl( CryptoSystem cryptoSystem,CryptoTokenPrivateKeyData privateKeyData);`

Enable decryption of unprocessed data.	> Perform one of the following steps:
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, create a method that performs decryption of unprocessed data, for example:
	`public void decryptRSA(RSACryptoSystem cryptoSystem,`
	`CryptoTokenPrivateKeyData privateKeyData,byte[] input, int inputOffset,`
	`byte[] output, int outputOffset)throws CryptoTokenException`
	`{`
	`try {//signDecryptHelper is a private helper method.`
	`signDecryptHelper(cryptoSystem, privateKeyData, input, inputOffset, output, outputOffset, DECRYPT_DESC, SmartCardSession.DECRYPT_OPERATION);`
	`}`
	`catch (CryptoUnsupportedOperationException e) {`
	`throw new CryptoTokenException(e.toString());`
	`}`
	`}`
	• To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, create a method that performs decryption of unprocessed data, for example:
	`public void decryptRSASmartCardImplCryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData, byte[] input, int inputOffset, byte[] output,int outputOffset)`

Task	Steps
Enable signing of unprocessed data.	> Perform one of the following steps: • To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.2 or later, create a method that signs unprocessed data, for example: `public void signRSA(RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData, byte[] input, int inputOffset,byte[] output, int outputOffset) throws CryptoTokenException, CryptoUnsupportedOperationException { signDecryptHelper(cryptoSystem, privateKeyData, input, inputOffset, output, outputOffset, SIGN_DESC, SmartCardSession.SIGN_OPERATION); }` • To create a cryptographic smart card driver that is compatible with BlackBerry Device Software Version 4.1 and Version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, create a method that signs unprocessed data, for example: `public void signRSASmartCardImpl (CryptoSystem cryptoSystem, CryptoTokenPrivateKeyData privateKeyData, byte[] input,int inputOffset, byte[] output, int outputOffset)`

See "Code sample: Enabling a CryptoToken object for RSA operations using a private key" on page 24 for more information.

Store the location of the private key file

Even though the private key file is stored on the smart card, the BlackBerry device needs to know that a private key file exists for a certificate. A class that implements the `CryptoTokenPrivateKeyData` interface can act as a pointer to a private key file on the smart card.

Task	Steps
Store the location of the private key file on the smart card.	> Implement the `CryptoTokenPrivateKeyData` interface.

Associate the implementing class object with the smart card that contains the private key file.	1. Create an instance variable for storing the smart card ID.
	`private SmartCardID _id;`
	2. Create an instance variable for storing the location of the private key file on the smart card.
	`private byte _file;`
	3. Create a constructor that associates an object from the class that implements a `PrivateKeyData` interface with the smart card.
	`public MyCryptoTokenData(SmartCardID id, byte file) { _id = id; _file = file; }`

Retrieve the ID of the smart card.	> Create a method that returns the `SmartCardID` instance variable. `public SmartCardID getSmartCardID() { return _id; }`

Task	Steps

Retrieve the location of the private key file on the smart card.	> Create a method that returns the location of the private key file on the smart card.
	`public byte getFile()`
	`{`
	`  return _file;`
	`}`

See "Code sample: Storing the location of a private key file on the smart card" on page 28 for more information.

2

Testing a cryptographic smart card driver

Set up the BlackBerry Device Simulator to test a cryptographic smart card driver
Set up a BlackBerry device to test a cryptographic smart card driver
Test the cryptographic smart card driver

Set up the BlackBerry Device Simulator to test a cryptographic smart card driver

To test a cryptographic smart card driver with the BlackBerry® Device Simulator, you require the Casira® End Point. Visit Cambridge Silicon Radio Lt. at www.btdesigner.com/devcasira.htm for more information.

Note: You do not require the Casira® Bluetooth® hardware and software development system.

1. In the BlackBerry Java Development Environment, on the taskbar, click **Start** > **Programs** > **Research In Motion** > **BlackBerry JDE 4.3.0** > **JDE**.
2. On the main menu, click **Edit** > **Preferences**.
3. Click the **Simulator** tab.
4. Click the **Ports** tab.
5. In the **Bluetooth test board port** field, type the port information.
6. Click **OK**.

Set up a BlackBerry device to test a cryptographic smart card driver

Task	Steps
Connect the BlackBerry IDE to a BlackBerry device using a USB port.	1. Install BlackBerry® Desktop Software Version 3.5.1 or later. 2. From the BlackBerry Developer Zone (http://na.blackberry.com/eng/developers/), download the **.debug** files that correspond to the version of the BlackBerry Device Software on the BlackBerry device. 3. Copy the **.debug** files to a folder on your computer. 4. In the BlackBerry IDE, specify the location of the **.debug** files. See the *BlackBerry Integrated Development Online Help* for more information about working with **.debug** files. 5. In the BlackBerry IDE, on the **Debug** menu, click **Attach to** > **Handheld** > **USB (*PIN*)**, where *PIN* is the PIN of the BlackBerry device.

Task	Steps
Set up to test signing and decrypting email messages with the S/MIME Support Package for BlackBerry smartphones, on a BlackBerry device with S/MIME support.	The following steps require a BlackBerry® Smart Card Reader. 1. Install the S/MIME Support Package for BlackBerry smartphones. Client access licenses are available from wireless service providers. For additional information on how to obtain the client access licenses, send an email message to government@blackberry.net. See the *S/MIME Support Package User Guide Supplement* for more information about installing the S/MIME Support package for BlackBerry smartphones on your computer or a BlackBerry device. 2. Connect the BlackBerry device to the computer. 3. At a command prompt, switch to the BlackBerry Java Development Environment bin folder. 4. Type the following command: JavaLoader [-usb] [-w*password*] load <file> • *password*: If a password is set, the password for the BlackBerry device • *file*: `The` .cod file that the cryptographic smart card driver downloads to the BlackBerry device

See the *BlackBerry Integrated Development Environment Help* for more information about testing a BlackBerry Java Application using the BlackBerry Integrated Development Environment.

Test the cryptographic smart card driver

Task	Steps
Make sure the cryptographic smart card driver is installed on a BlackBerry device.	1. On the BlackBerry device, click **Options** > **Security Options** > **Smart Card**. 2. Ensure the cryptographic smart card appears in the **Registered Card Drivers** section.
Make sure the options display for the cryptographic smart card driver.	If you implemented `isDisplaySettingsAvailableImpl()` to return true. 1. On the BlackBerry device, click **Options** > **Security Options** > **Smart Card**. 2. In the **Registered Card Drivers** section select the cryptographic smart card driver you are testing. 3. Click **Driver Settings**.
Test signing and decrypting email messages with the S/MIME Support Package for BlackBerry smartphones on a BlackBerry device with S/MIME support.	1. On the BlackBerry device, click **Options** > **Security Options** > **Certificates**. 2. Click **Import Smart Card Certs**. 3. If required, type your PIN. 4. Select the certificate you want to import from the smart card. 5. Type the smart card password. 6. Specify S/MIME to sign and decrypt email messages. See *S/MIME Support Package User Guide Supplement* for more information about signing, sending, and decrypting email messages.
Enable and test two-factor authentication on a BlackBerry device.	1. On the BlackBerry device, click **Options** > **Security Options** > **General Settings**. 2. Verify that your smart card is inserted in the BlackBerry Smart Card Reader. 3. Change the **User Authenticator Password** field to **Enabled**. 4. Click **Save**. 5. When prompted, type the smart card password. 6. Click **Enter**.

3

Code samples

Code sample: Creating a cryptographic smart card object
Code sample: Creating a cryptographic session for a cryptographic smart card driver
Code sample: Enabling a CryptoToken object for RSA operations using a private key
Code sample: Storing the location of a private key file on the smart card

Code sample: Creating a cryptographic smart card object

Example: MyCryptoSmartCard.java

```
/**
* MyCryptoSmartCard.java
* Copyright (C) 2001-2007 Research In Motion Limited. All rights reserved.
*/
package com.rim.samples.device.smartcard;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;
import net.rim.device.api.crypto.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.system.*;


/**
* This class represents a kind (or model or family) of a physical smart card.
* There should only be one instance of this class in the system at one time. The instance
* is managed by the SmartCardFactory.
*/
public class MyCryptoSmartCard extends CryptoSmartCard implements Persistable
{
    private final static byte MY_ATR [] = { (byte)0x3b, (byte)0x7d, (byte)0x11,
   (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x31,
   (byte)0x80, (byte)0x71, (byte)0x8e, (byte)0x64,
   (byte)0x86, (byte)0xd6, (byte)0x01, (byte)0x00,
   (byte)0x81, (byte)0x90, (byte)0x00 };

   private final static AnswerToReset _myATR = new AnswerToReset( MY_ATR );
   private static final String LABEL = “RIM Sample”;
   private static final String DISPLAY_SETTINGS = “Show driver properties/settings now”;
   private static final String RSA = “RSA”;


   /**
   * This method is invoked when the BlackBerry device starts and registers this
   * cryptographic smart card driver with the smart card factory.
   * Registering this cryptographic smart card driver with the smart card factory
   * automatically registers the cryptographic smart card driver with the user
   * authenticator framework which allows the smart card to be used as a second factor of
```

```
* authentication for unlocking a BlackBerry device.
* In the BlackBerry Integrated Development Environment, in the settings for the smart
* card driver project, make sure the project type is set to Library.
* Make sure to select the auto-run on startup setting.
*/
public static void libMain( String args[] )
{
try{
    SmartCardFactory.addSmartCard( new MyCryptoSmartCard() );}
    catch(ControlledAccessException cae)
    {
// Application control may not allow your driver to be used with the user authenticator
// framework, in which case it will throw a ControlledAccessException.
// Your driver was registered with the smart card API framework and can still
// be used for importing certificates and signing/decrypting messages.
    }
}


 /**
 * Retrieve the session handler for this smart card.
 * Implementations of this method should not bring up UI.
 */
protected SmartCardSession openSessionImpl( SmartCardReaderSession readerSession )
throws SmartCardException {
    return new MyCryptoSmartCardSession( this, readerSession );
 }

/**
* Determine if the file system should use this smart card object
* to communicate with a physical smart card that has the given AnswerToReset.
* The system invokes this method to ascertain which smart card implementation it should
* use to communicate with a physical smart card found in a BlackBerry Smart Card Reader.
*/
protected boolean checkAnswerToResetImpl( AnswerToReset atr )
{
// If this method returns false, the cryptographic smart card driver will not be used to
// perform additional operations on a particular smart card.
// This method should only return true if you support the particular ATR. If this method
// returns true when there is no support for the smart card that corresponds to the ATR,
// this may prevent other cryptographic smart card drivers from functioning correctly.
// The AnswerToReset parameter contains the full ATR from the smart card.
    // Your implementation of this method may check the entire ATR or just parts of
    // the ATR, as long as the cryptographic smart card driver supports the
    corresponding smart card.

    return _myATR.equals( atr );
}


/**
* Retrieve a label associated with this smart card.
* The string should not include the words “smart card”, as the file system uses this
* this method to generate strings such as “Please insert your smart card”.
*/
protected String getLabelImpl()
{
    return LABEL;
}
```

```
/**
* Retrieves this smart card’s capabilities
*/
protected SmartCardCapabilities getCapabilitiesImpl()
{
   return new SmartCardCapabilities( SmartCardCapabilities.PROTOCOL_T0 );
}


/**
* Determine if this smart card can display its settings.
*/
protected boolean isDisplaySettingsAvailableImpl( Object context )
{
   return true;
}


/**
* Display this smart card’s settings.
* This method will be invoked from the smart card options screen when
* the user selects the driver and chooses to view the settings of that driver.
*
* This method could be called from the event thread. The driver should not block
* the event thread for long periods of time.
*
* @param context Reserved for future use.
*/
protected void displaySettingsImpl( Object context )
{
   Dialog.alert( DISPLAY_SETTINGS );
}


/** Retrieve the algorithms supported by this smart card.
*
* @return one or more of “RSA”, “DSA”, or “ECC”
*/
public String[] getAlgorithms()
{
   return new String [] { RSA };
}


/** Retrieve a crypto token that supports the given algorithm.
* @param algorithm Name of the algorithm.
*
* @return Crypto Token supporting the named algorithm.
*
* @throws NoSuchAlgorithmException If the specified algorithm is invalid.
* @throws CryptoTokenException If there is a token-related problem.
*/
public CryptoToken getCryptoToken( String algorithm ) throws NoSuchAlgorithmException,
CryptoTokenException
{
   if ( algorithm.equals( RSA ) ) {
```

```
            return new MyRSACryptoToken();
            }

        throw new NoSuchAlgorithmException();
    }
}
```

Code sample: Creating a cryptographic session for a cryptographic smart card driver

Example: MyCryptoSmartCardSession.java

```
/**
* MyCryptoSmartCardSession.java
* Copyright (C) 2001-2007 Research In Motion Limited. All rights reserved.
*/

package com.rim.samples.device.smartcard;

import net.rim.device.api.crypto.*;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.certificate.x509.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;

/**
* This class represents a communication session with a physical smart card.
*
* Over this session, Application Protocol Data Units may be exchanged with the smart card
* to provide the desired functionality.
* Do not hold open sessions when not using them; they should be short-lived.
* As a security precaution, only one open session is allowed to exist per SmartCardReader;
* subsequent openSession() requests will block until the current session is closed.
*/
public class MyCryptoSmartCardSession extends CryptoSmartCardSession
{
    // We assume that the smart card has three certificates identified by: ID_PKI,
    SIGNING_PKI
    // and ENCRYPION_PKI. Your particular smart card may have a different number of
    // certificates or be identified differently. These three certificates are merely an
    example of
    // what a smart card might contain.

    public static final byte ID_PKI = (byte)0x00;
    public static final byte SIGNING_PKI = (byte)0x01;
    public static final byte ENCRYPTION_PKI = (byte)0x02;
    private static final String WAITING_MSG = “Please Wait”;
    private static final String ID_STRING = “Jason Hood”;
    private static final String ID_CERT = “ID Certificate”;
    private static final String SIGNING_CERT = “Signing Certificate”;
    private static final String ENCRYPTION_CERT = “Encryption Certificate”;
```

```
/**
* Construct a new MyCryptoSmartCardSession object.
*
* @param smartCard Smart card associated with this session
* @param readerSession Reader session commands sent to this smart card.
*/
protected MyCryptoSmartCardSession( SmartCard smartCard, SmartCardReaderSession
readerSession )
{
   super( smartCard, readerSession );
}


/**
* Close this cryptographic smart card session.
*
* Implementations should not close the underlying SmartCardReaderSession. Use this
* method for cleaning up the session prior to its closure.
*/
protected void closeImpl()
{
   // Do any session cleanup needed here.
}

/**
* Retrieve the maximum number of allowed login attempts.
* The method returns Integer.MAX_VALUE if an infinite number of attempts are allowed.
*/
protected int getMaxLoginAttemptsImpl() throws SmartCardException
{
   return 5;
}


/**
* Retrieve the remaining number of login attempts allowed (before the smart card will
* lock, or Integer.MAX_VALUE if the smart card will not lock.)
*/
protected int getRemainingLoginAttemptsImpl() throws SmartCardException
{
   return 4;
}


/**
* Log into the smart card with the given password.
* This method should not bring up the UI.
*/
protected boolean loginImpl( String password ) throws SmartCardException
{
   // Create a CommandAPDU which your smart card will understand
   CommandAPDU command = new CommandAPDU( (byte)0x00, (byte)0x20, (byte)0x00,
   (byte)0x00 );
   command.setLcData( password.getBytes() );
   ResponseAPDU response = new ResponseAPDU();
   sendAPDU( command, response );
```

```
        // Check for response codes specific to your smart card
        if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
            return true;
        }
        else if ( response.checkStatusWords( (byte)0x64, (byte)0xF8 ) ) {
            throw new SmartCardLockedException();
        }
        else {
            // Authentication failed
            return false;
            }
    }

    /**
    * Retrieve an ID for this session's associated smart card.
    */
    protected SmartCardID getSmartCardIDImpl() throws SmartCardException
    {
        // Retrieve a unique ID from the card
        byte [] uniqueCardData = new byte[] { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b };

        // Convert byte array to a long
        SHA1Digest digest = new SHA1Digest();
        digest.update( uniqueCardData );
        long idLong = byteArrayToLong( Arrays.copy( digest.getDigest(), 0, 8 ) );

        // Using friendly display name
        return new SmartCardID( idLong , ID_STRING, getSmartCard() );
    }



    /**
    * Converts <code>array</code> into a long integer (Note: the returned value should
    * be considered as an unsigned value).
    * @throws IllegalArgumentException if <code>array</code> contains a number bigger
    * than 64 bits.
    *
    * Note:
    * If your cryptographic smart card driver is only designed to work with
    * BlackBerry Version 4.2 or later, you can replace this method with a call to
    * CryptoByteArrayArithmetic.valueOf( byte [] ).
    *
    */
    private long byteArrayToLong( byte[] array )
    {
        if ( array == null ) {
            throw new IllegalArgumentException();
        }

        // Remove the leading zeros from given byte[] and returns a new byte[] without them.
        int zeros = 0;
        for ( int i = 0; i < array.length && array[i] == 0; i++ ){
            zeros++;
        }
```

```
   if ( zeros != 0 ) {
      array = Arrays.copy( array, zeros, array.length - zeros );
   }

   int length = array.length;
   if( length > 8 ) {
      throw new IllegalArgumentException();
   }

   long n = 0;

   for( int i=0; i<length; i++ ) {
      n <<= 8;
      n += array[i] & 0xff;
   }

   return n;
}



/**
* Retrieve some random data from the smart cards internal Random Number Generator.
*/
protected byte [] getRandomBytesImpl( int maxBytes ) throws SmartCardException
{
   // Create a CommandAPDU which your smart card will understand
   CommandAPDU command = new CommandAPDU( (byte)0x00, (byte)0x4C, (byte)0x00,
   (byte)0x00, maxBytes );

   ResponseAPDU response = new ResponseAPDU();
   sendAPDU( command, response );

   // Check for response codes specific to your smart card
   if( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
      // The appropriate response code containing the random data
      return response.getData();
   }

   return null;
}


/**
* Retrieve certificates from the card.
*
* @return An array of certificates which are present on the card.
*/
protected CryptoSmartCardKeyStoreData[] getKeyStoreDataArrayImpl() throws
SmartCardException, CryptoTokenException
{
   try {
      // Show a progress dialog to the user as this operation may take a long time.
      displayProgressDialog( WAITING_MSG, 4 );
      // The certificates need to be associated with a particular card.
      SmartCardID smartCardID = getSmartCardID();
```

```
        RSACryptoToken token = new MyRSACryptoToken();
        RSACryptoSystem cryptoSystem = new RSACryptoSystem( token, 1024 );
        RSAPrivateKey privateKey;
        CryptoSmartCardKeyStoreData[] keyStoreDataArray = new
        CryptoSmartCardKeyStoreData[ 3 ];

        // This encoding would be extracted from the card using a series of APDU
        commands.
        Certificate certificate = null;

        // Extract the certificate encoding from the card.
        byte [] certificateEncoding = new byte[0];

        try {
           certificate = new X509Certificate( certificateEncoding );
           }
           catch( CertificateParsingException e )
           { // invalid X509 certificate }
    }

        stepProgressDialog( 1 );

        privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
        smartCardID, ID_PKI ) );
        keyStoreDataArray[ 0 ] = new CryptoSmartCardKeyStoreData( null, ID_CERT,
        privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );
        stepProgressDialog( 1 );

        privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
        smartCardID, SIGNING_PKI ) );
        keyStoreDataArray[ 1 ] = new CryptoSmartCardKeyStoreData( null, SIGNING_CERT,
        privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );

        stepProgressDialog( 1 );

        privateKey = new RSAPrivateKey( cryptoSystem, new MyCryptoTokenData(
        smartCardID, ENCRYPTION_PKI ) );
        keyStoreDataArray[ 2 ] = new CryptoSmartCardKeyStoreData( null, ENCRYPTION_CERT,
        privateKey, null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );
        stepProgressDialog( 1 );

        // Sleep so the user sees the last step of the progress dialog move to 100%
        try {Thread.sleep( 250 );
           } catch ( InterruptedException e ) {
           }

        dismissProgressDialog();
        return keyStoreDataArray;

        } catch ( CryptoUnsupportedOperationException e ) {
        } catch ( UnsupportedCryptoSystemException e ) {
    } catch ( CryptoTokenException e ) {
    }


    throw new SmartCardException();
}
```

```
    /**
    * Send some data to the smart card for signing or decryption.
    */
    /*package*/ void signDecrypt( RSACryptoSystem cryptoSystem,
    MyCryptoTokenData privateKeyData,byte[] input, int inputOffset,
    byte[] output, int outputOffset ) throws SmartCardException
    {
       // Check for nulls
       if ( cryptoSystem == null || privateKeyData == null || input == null || output ==
       null) {
          throw new IllegalArgumentException();
       }


       // Validate the input parameters
       int modulusLength = cryptoSystem.getModulusLength();

       if ( ( input.length < inputOffset + modulusLength ) || ( output.length <
       outputOffset + modulusLength ) ) {
          throw new IllegalArgumentException();
       }

       // Construct the response Application Protocol Data Unit
       ResponseAPDU response = new ResponseAPDU();


       // Construct the command and set its information
       // Create a CommandAPDU which your smart card will understand
       CommandAPDU signAPDU = new CommandAPDU( (byte)0x80, (byte)0x56, (byte)0x00,
       (byte)0x00, modulusLength );
       signAPDU.setLcData( input, inputOffset, input.length - inputOffset );

       // Send the command to the smart card
       sendAPDU( signAPDU, response );


       // Validate the status words of the response
       // Check for response codes specific to your smart card
       if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
          byte [] responseData = response.getData();
          System.arraycopy( responseData, 0, output, outputOffset, responseData.length );
       }
       else {
          throw new SmartCardException( “Invalid response code, sw1=” +
          Integer.toHexString( response.getSW1() & 0xff ) + “ sw2=” + Integer.toHexString(
          response.getSW2() & 0xff ) );
       }
    }

}
```

Code sample: Enabling a CryptoToken object for RSA operations using a private key

Example: MyRSACryptoToken.java

```
/**
* MyRSACryptoToken.java
* Copyright (C) 2001-2007 Research In Motion Limited. All rights reserved.
*/
package com.rim.samples.device.smartcard;

import net.rim.device.api.smartcard.*;
import net.rim.device.api.crypto.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.util.*;
import net.rim.device.api.crypto.certificate.x509.*;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.asn1.*;
import net.rim.device.api.compress.*;
import net.rim.device.api.i18n.*;

/**
* This class describes an implmentation of an RSA cryptographic token.
*
* The RIM Crypto API will use this object to perform a private key RSA operation.
* This object should delegate the operation to the smart card.
*/
final class MyRSACryptoToken extends RSACryptoToken implements Persistable
{
    private static final String DECRYPT_DESC = “The private key will be used to decrypt
    encrypted data.”;
    private static final String SIGN_DESC = “The private key will be used to generate a
    digital signature.”;

    /**
    * Constructs a new MyRSACryptoToken object.
    */
    MyRSACryptoToken()
    {
    }


    /**
    * Determine if this token does the user authentication for the system.
    *
    * If not the KeyStore will prompt for the key store password when the user
    * tries to access the private key.
    *
    * @return True if this token will prompt for the necessary
    * user authentication when private key access is requested.
    */
    public boolean providesUserAuthentication()
    {
       return true;
    }
```

```
/**
* Determine if this token supports the chosen operation using the provided system.
*
* @param cryptoSystem Crypto System to check against.
* @param operation Operation to check: either KEY_GENERATION,
* PUBLIC_KEY_OPERATION, PRIVATE_KEY_OPERATION,* or some other value
* specific to the crypto system that indicates the operation to check.
*/
public boolean isSupported( CryptoSystem cryptoSystem, int operation )
{
   return ( operation == PRIVATE_KEY_OPERATION );
}


/**
* Determines if the given key and crypto system
* support RSA encryption.
*
* @return True if the token supports RSA encryption.
*
* @param cryptoSystem Crypto system to check.
* @param privateKeyData Private key data.
*
* @throws CryptoTokenException If an error occurs with a crypto
* token or the crypto token is invalid.
*/
public boolean isSupportedDecryptRSA( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData )throws CryptoTokenException
{
   return privateKeyData instanceof MyCryptoTokenData;
}


/**
* Perform RSA decryption of unprocessed data.
*
* <p>
* Notes: To reveal the plaintext bytes, the RSACryptoToken should perform a raw RSA
* private key operation on the input data.
* The plaintext bytes are typically padded with the type of padding dependent on the
* application that encrypted the data.
*
* Typically Public-Key Cryptography Standards (PKCS) #1 version 2.0 is used to pad the
data, however other schemes
* such as OAEP can also be used.
*
* If the RSACryptoToken removes the padding in the input data, this method must re-add
* the same type of padding before the method completes its operations.
*
* Data encrypted using the BlackBerry S/MIME implementation currently uses Public-Key
Cryptography Standards (PKCS) #1
* padding but may use other padding methods in the future.
* <p>
*
* @param cryptoSystem Crypto system associated with the token.
* @param privateKeyData RSA private key.
```

```
* @param input Input data.
* @param inputOffset First byte of the input data to read.
* @param output Buffer for the output data.
* @param outputOffset Position in the output buffer to receive the first written byte.
*
* @throws CryptoTokenException Thrown if an error occurs with a crypto
* token or the crypto token is invalid.
*/
public void decryptRSA( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData,byte[] input, int inputOffset,
   byte[] output, int outputOffset )throws CryptoTokenException
{
   try {
      signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output,
      outputOffset, DECRYPT_DESC, SmartCardSession.DECRYPT_OPERATION );
      }
   catch ( CryptoUnsupportedOperationException e ) {
   throw new CryptoTokenException( e.toString() );
   }
}


/**
* Perform RSA signing of unprocessed data.
*
* <p>
* Notes: The RSA Crypto token should perform a raw RSA private key operation on the
* input data. The input data is typically padded, with the type of padding dependent on
* the application that requested the signature.  Typically Public-Key Cryptography
Standards (PKCS) #1 version 2.0
* is used to pad the data, however other schemes such as Probabilistic Signature Scheme
(PSS) or ANSI X9.31 could also
* be used.
*
* If the RSA Crypto token requires the padding to be removed before signing, this method
* will need to detect and remove the type of padding that is currently used.  The RSA
* Crypto token should only re-apply the same type of padding that was originally applied
* to the data. If the RSA Crypto token is unable to re-apply the same type of padding,
* this method should throw an UnsupportedOperationException.
* Signature requests which come from BlackBerry's S/MIME implementation currently use
* Public-Key Cryptography Standards (PKCS) #1 padding but may use other padding methods
in the future.
* <p>
* @param cryptoSystem Cypto system associated with the token.
* @param privateKeyData RSA private key.
* @param input Input data.
* @param inputOffset First byte of the input data to read.
* @param output The buffer for the output data.
* @param outputOffset Position in the output buffer to receive the first written byte.
* @throws CryptoTokenException Thrown if an error occurs with the crypto
* token or the crypto token is invalid.
* @throws CryptoUnsupportedOperationException Thrown if a call is made to an
* unsupported operation or if the token does not support signing due to the type of
* padding around the encoded message.
*/
public void signRSA(    RSACryptoSystem cryptoSystem, CryptoTokenPrivateKeyData
privateKeyData, byte[] input, int inputOffset,byte[] output, int outputOffset )
throws CryptoTokenException, CryptoUnsupportedOperationException
```

```
{
   signDecryptHelper( cryptoSystem, privateKeyData, input, inputOffset, output,
   outputOffset, SIGN_DESC, SmartCardSession.SIGN_OPERATION );
}


/**
* Help signing and decryption operations.
* This helper method assists data signing and decryption because
* the operations are very similar.
*/
private void signDecryptHelper( RSACryptoSystem cryptoSystem,
CryptoTokenPrivateKeyData privateKeyData, byte[] input, int inputOffset,
byte[] output, int outputOffset,String accessReason,int operation )
throws CryptoTokenException, CryptoUnsupportedOperationException
{
   SmartCardSession smartCardSession = null;

   try {

      if( privateKeyData instanceof MyCryptoTokenData ) {

      SmartCardID smartCardID = ((MyCryptoTokenData) privateKeyData
      ).getSmartCardID();

      smartCardSession = SmartCardFactory.getSmartCardSession( smartCardID );

      if ( smartCardSession instanceof MyCryptoSmartCardSession ) {
         MyCryptoSmartCardSession mySmartCardSession = ( MyCryptoSmartCardSession
         )smartCardSession;

      // We must provide the user authentication since we returned true from
      //providesUserAuthentication()
      // Also, the smart card PIN is required for private key access.
         mySmartCardSession.loginPrompt( accessReason, operation );
         mySmartCardSession.signDecrypt( cryptoSystem,
         (MyCryptoTokenData)privateKeyData, input, inputOffset, output, outputOffset
         );

      return;
      }
   }


        throw new RuntimeException();

      } catch ( SmartCardSessionClosedException e ) {
         throw new CryptoTokenCancelException( e.toString() );
      } catch ( SmartCardCancelException e ) {
         throw new CryptoTokenCancelException( e.toString() );
      } catch( SmartCardRemovedException e ) {
         throw new CryptoTokenCancelException( e.toString() );
      } catch ( SmartCardException e ) {
         throw new CryptoTokenException( e.toString() );
      } finally {
         if ( smartCardSession != null ) {
         smartCardSession.close();
         }
```

```
            }
    }
}
```

Code sample: Storing the location of a private key file on the smart card

Example: MyCryptoTokenData.java

```
/**
* MyCryptoTokenData.java
* Copyright (C) 2001-2007 Research In Motion Limited. All rights reserved.
*/

package com.rim.samples.device.smartcard;

import net.rim.device.api.crypto.*;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;


/**
* This class stores the location of the private key file on the smart card.
*
*/
final class MyCryptoTokenData implements CryptoTokenPrivateKeyData, Persistable
{
    /**
    * Smart card containing the private key.
    */
    private SmartCardID _id;


    /**
    * Location of the private key file on the smart card.
    */
    private byte _file;


    /**
    * Constructs a new MyCryptoTokenData object
    *
    * @param id ID of the smart card containing the private key file
    * @param file Location of the private key file.
    */
    public MyCryptoTokenData( SmartCardID id, byte file )
    {
        _id = id;
        _file = file;
    }
```

```
    /**
    * Retrieve the ID of the key file containing the private key file.
    *
    * @return ID of the smart card.
    */
    public SmartCardID getSmartCardID()
    {
       return _id;
    }



    /**
    * Retrieve the location of the private key file on the smart card.
    *
    * @return Location of the private key file.
    */
    public byte getFile()
    {
       return _file;
    }

}
```

©2007 Research In Motion Limited

Published in Canada.